C25

BEST AVAILABLE COPY

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 810 524 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
03.12.1997 Bulletin 1997/49

(51) Int Cl.6 G06F 9/46

(21) Application number: 97303576.9

(22) Date of filing: 27.05.1997

(84) Designated Contracting States:
DE FR GB SE

(30) Priority: 30.05.1996 US 657712
25.11.1996 US 756325
28.03.1997 US 828318

(71) Applicant: SUN MICROSYSTEMS, INC.
Mountain View, California 94043-1100 (US)

(72) Inventors:
- Gosling James A
Woodside California 94062 (US)
- Diwanji Pavani
San Jose California 95126 (US)
- Connelly David W.
Los Altos California 94022 (US)

(74) Representative:
Cross, Rupert Edward Blount et al
BOULT WADE TENNANT,
27 Furnival Street
London EC4A 1PQ (GB)

(54) Apparatus and method for processing servlets

(57) A method and apparatus for operating a local server computer of a client-server network includes a technique to receive a request from a client computer of the client-server network. A determination is made whether the request requires dynamically generated information from a servlet object of the client-server network. If so, a specified servlet object corresponding to the request may be uploaded from a remote server computer of the client-server network. The specified servlet object is then executed to obtain dynamically generated information corresponding to the request.

EP 0 810 524 A1

Printed by Jouve 75001 PARIS (FR)

Description

Brief Description of the Invention

This invention relates generally to exchanging information in a client-server computer environment. More particularly this invention relates to an improved technique for responding to information requests at a server computer.

Background of the Invention

Client-server computer networks are well known. The most prominent example of a client-server computer network is the World Wide Web of computers. In a client-server computer network a server computer receives a request for information from a client computer. Web server software operating on the server computer typically retrieves the requested information from a file stored on a permanent storage device and transmits the file over the network to the client computer that requested the information. The web server software is generally not written using an object oriented programming language. Thus, it is not easily extended to provide new functionality. Given the dynamic nature of today's software marketplace, a product's lack of flexibility and extendibility can seriously hinder the marketability of the product.

Current web server software can generate a file dynamically in response to a request from a client computer. Typically, the web server receives the request and then forks a Common Gateway Interface (CGI) process to dynamically create the file. Once the file has been created, the web server software transmits the file back to the client computer. Unfortunately, it is computationally expensive to fork a process each time dynamic information needs to be generated.

In view of the foregoing, it would be highly desirable to provide a web server which dynamically generates information in response to a client computer request, but which does not incur a process start-up expense while generating the dynamic information. Further, it would be highly desirable to provide an object oriented web server environment that is flexible and extendible.

Summary of the Invention

The invention includes a method and apparatus for operating a local server computer of a client-server network. The invention includes a technique to receive a request from a client computer of the client-server network. A determination is made whether the request requires dynamically generated information from a servlet object of the client-server network. If so, a specified servlet object corresponding to the request may be uploaded from a remote server computer of the client-server network. The specified servlet object is then executed to obtain dynamically generated information corresponding to the request.

The servlet objects of the invention provide an object oriented web server environment which is flexible and extendible. The client-server network of the invention is populated with the servlet objects. The servlet objects operate in a continual loop until invoked. Thus, there is no startup overhead associated with execution of the servlet objects. By observing a common applications program interface, the servlet objects can run in any server environment. A feature of the invention allows untrusted servlet objects to be executed in a secure area, with the dynamically generated information being passed from the secure area into the remaining server environment.

Brief Description of the Drawings

Examples of the invention will now be described in conjunction with the accompanying drawings, in which:

FIGURE 1 illustrates a client-server computer network in accordance with an embodiment of the invention.

FIGURE 2 is a simplified illustration of the interactions between a web server and the servlets.

FIGURE 3 is a simplified illustration of the interactions between a web server and a servlet loaded from an external server.

FIGURE 4 illustrates processing steps associated with a servlet processing routine.

FIGURE 5 illustrates processing steps associated with a servlet processing routine.

Like reference numerals refer to corresponding parts throughout the several views of the drawings.

Detailed Description of the Invention

Figure 1 illustrates a client-server computer network 20.

The network 20 includes at least one client computer 22 and at least one server computer 24. The client computer 22 and the server computer 24 are connected by a transmission channel 26, which may be any wire or wireless transmission channel.

The client computer 22 is a standard computer including a Central Processing Unit (CPU) 30 connected to a memory (primary and/or secondary) 32. The memory 32 stores a number of computer programs, including a "browser" 34. As known in the art, a browser is used to communicate with remote server computers 24 and to visually present the information received from such computers. The client computer 22 establishes network communications through a standard network connection device 36.

The server computer 24 includes standard server computer components, including a network connection device 40, a CPU 42, and a memory (primary and/or secondary) 44. The memory 44 stores a set of computer programs to implement the processing associated with the invention. The memory 44 stores a web server 46. The web server 46 may be of the type known in the art, which is modified to include the additional programs shown in Figure 1. That is, in an embodiment of the invention, a standard web server 46 is modified to include a server acceptor thread 48, a connection queue 50, a pool administrator 52, a thread pool 54, servlets 56, a servlet map 58, a security administrator 60, and boundary servlets 62.

Figure 2 is a simplified illustration of a server computer 24A constructed in accordance with an embodiment of the invention. The figure shows a web server 46 interacting with a set of servlets 56A-56N. In particular, the web server 46 interacts with the servlets through an application program interface (API). As indicated in Figure 1, the web server 46 and the servlets 56 are stored in memory 44. The web server 46 may be standard web server software that is modified to include the functionality described herein. Each servlet 56 is a piece of software code which is used to dynamically generate information. Each servlet 56 is an instantiated software object waiting to be invoked. Once it is invoked, it dynamically generates information. Note that this technique of dynamically generating information is distinct from the typical process of fetching static information from a permanent storage device. The technique of the invention is similar to a CGI script in the sense that it dynamically generates information. However, unlike a CGI script, a servlet object of the present invention is instantiated at server start-up. Thus, the servlet can be thought of as operating in a continual loop waiting to be executed. Observe that after instantiation there is no computational start-up expense when the servlet is called.

Figure 3 is a general illustration demonstrating additional features of the invention. Figure 3 illustrates a local server computer 24A which receives a request from a client computer (not shown) over transmission channel 26. The web server 46 determines that dynamically generated information from a servlet object is required. In this case, the servlet object is not initially on the local server computer 24A, thus it is uploaded by the local server computer 24A from a remote server computer 24B using communication link 26. In the example of Figure 3, servlet 56P is passed from the remote server computer 24B to the local server computer 24A.

Figure 3 illustrates another feature of the invention. In particular, it illustrates that the uploaded servlet 56P is executed in a security area 57 of the local server computer 24A. After execution, the results are passed to a boundary servlet 60 in the remaining portion of the local server computer 24A. This security feature allows untrusted servlets to be safely executed.

The foregoing discussion provides a general description of the features and benefits of the invention. Attention now turns to a more detailed description of these features and benefits. The left side of Figure 4 illustrates processing steps associated with an embodiment of the invention. The right side of Figure 4 illustrates program components that may be used to execute these operations.

The first processing step shown in Figure 4 is to determine whether a new request has been received (step 70). As indicated above, a request is a request for information from a client computer 22 to a server computer 24. The operation of a client computer 22 requesting information from a server computer 24 is well known. It is typically performed using a Uniform Resource Locator or URL. A URL specifies a computer and a file. A typical URL is http://SU/123. This URL is an instruction to retrieve the file "123" from the State University computer "SU" using the Hypertext Transfer Protocol "HTTP".

As shown in Figure 4, a server acceptor thread 48 is used to process each new request. Preferably, the invention is implemented as a connection-oriented web server with a server acceptor thread that continually loops while accepting requests. Once a request is received, it dispatches the request to a connection queue (step 72). As shown in Figure 1, the connection queue 50 is formed in the memory of the local server computer 24.

If no new request is received, then a check is made to determine whether the queue is empty (step 71). If the queue is not empty or a new request has been received, processing proceeds to step 74. Step 74 entails thread pool administration operations, which are executed by a pool administrator 52. Figure 1 illustrates a thread pool 54. The thread pool 54 is a pool of threads that are used for request processing. Individual threads fetch and process requests from the connection queue 50. The pool administrator 52 operates to ensure that there is a thread for each request in the connection queue 50. The pool administrator 52 creates or forks additional threads to handle new requests in the connection queue 50. If a maximum number of threads is reached, the pool administrator 52 blocks new requests from entering the connection queue 50. In such a case, the server computer does not receive new requests. On the other hand, if a thread has been waiting more than a predetermined period of time for a request from the connection queue 50, then the pool administrator 60 will destroy it. Preferably, a new handler thread is created using the buffer space of

a destroyed handler thread. In other words, the invention is preferably implemented by using a specific buffer memory space for a thread. When a thread is destroyed, the buffer memory space is cleared, but it is assigned to a new thread. By reusing allocated memory in this manner, this embodiment of the invention minimizes the amount of memory used by the system, especially when compared to systems which allocate and deallocate memory on a per request basis.

After the thread pool administration operations are performed (step 74) a thread retrieves a request from the connection queue (step 76). The thread then maps the request to a servlet name (step 78). The servlet may be specified by a URL, in which case the mapping process is direct. On the other hand, some translation process may be required to identify which servlet will be able to service the request. The mapping operation may be performed in one of the following ways. A server administrator may specify that some kinds of client requests always map to a particular servlet. For example, one which talks to a particular database. A server administrator may specify that part of the client request is the name of the servlet, as found in an administered servlets directory. At many sites, that directory would be shared between servers which share the load of processing for the site's clients. Some servers may be able to automatically invoke servlets to filter the output of other servlets, based on their administrative configuration. For example, particular types of servlet output may trigger post-processing by other servlets, perhaps to perform format conversions. Properly authorized clients may specify the servlet to be invoked, without administrative intervention.

Security operations may also be performed by the thread (step 80). A security administrator 60 may be used to identify trusted and untrusted classes of servlets. The decision to trust a servlet may be established by a set of rules associated with the security administrator 60. For example, the security administrator 60 may decide to trust all local servlets and mistrust all uploaded network servlets. Untrusted servlets are then executed in the security area 57, as shown in Figure 3. The security administrator 60 may also be used to determine if the servlet is authorized to perform predetermined risky operations. Security information of this type may be stored in the thread.

JAVA servlets in accordance with the invention provide strong security policy support. This is because all JAVA environments provide a Security Manger which can be used to control whether actions such as network or file access are to be permitted. By default, all servlets are untrusted and are not allowed to perform operations such as accessing network services or local files. However, servlets "built into" the server, or servlets which have been digitally signed as they were put into JAVA Archive files, may be trusted and granted more permissions by the security manager. A digital signature on executable code indicates that the organization which signed the code "vouches for it" in some sense. Such signatures can't support accountability by themselves, but they do indicate a degree of assurance that may be placed on use of that code. For example, a particular signature from an MIS organization might be required on all code which is granted general access to network services within a corporate intranet. That signature might only be used on code which is strongly believed not to violate particular security policies. Extension APIs in other languages, such as C or scripting languages, can't support such fine grained access controls even if they do allow digital signatures for their code.

After security operations are performed (step 80), the thread is dispatched (step 82 of Figure 5). The dispatch operation entails invoking a servlet so that it generates the requested dynamic information. The dispatch operation is one of two types. A decision is made to determine whether the servlet is local (step 84). If the servlet is local, then the local servlet is executed (step 86). This results in the generation of dynamic information that is then processed by the web server 46 in a standard manner. The web server 46 typically passes the information back to the client computer using known techniques. The exchange of information between a servlet and the web server 46 is achieved through an application program interface, which is described below.

If the servlet is not local, then it is uploaded from a remote server 24B (step 88). A decision is then made regarding whether the uploaded servlet is safe (step 90). Recall that the security operation step resulted in the thread acquiring information regarding security parameters for servlets. If there are no security problems associated with the uploaded servlet, then it is executed locally (step 92). On the other hand, if a security problem is identified, then the servlet is executed in a security area (step 94). Thereafter, the dynamically generated results are passed to the non-security area (step 96). A boundary servlet may be used for this purpose. The boundary servlet may be implemented through the use of stubs and subcontracts or through other "fire wall" techniques known in the art. After the servlet is executed, processing returns to step 70 of Figure 4.

The operation of the invention has now been fully described. Attention now turns to a more particular discussion of the servlet objects that are used in accordance with the invention and an embodiment of the application program interface used in connection with the servlet objects. As indicated above, the servlet objects are objects are software objects that are used to dynamically generate information. They are instantiated objects that sit in a loop waiting to be invoked. Preferably, they are implemented as object bytecodes in the JAVA™ programming language. It is well known that the JAVA™ programming language is used to implement "applets" on a client computer. An "applet" is executable JAVA object bytecodes that are used to generate a graphical display on a client computer. The servlets embodying the present invention are executed on the server side and do not have graphical content.

A servlet is typically instantiated on server startup. In the alternative, the servlet may be instantiated under a predetermined set of conditions or by client invocation. The servlet may be instantiated and executed by using its URL

BNSDOCID: <EP 0810524A1>

(e g http //host/ < servlet URL>). The http protocol supports the passing of arguments. thus arguments may be passed to the servlet (e g http.//host/ < servlet URL>?< arguments >) The properties object is a JAVA programming language properties class which comprises a set of "name value" pairs A system administrator can pass arguments to an instantiated HttpServlet object through the properties object In this way the system administrator can "customize" an HttpServlet for a particular server at a particular site For example the system administrator can pass the Httpservlet object site specific information about the network location of a database which stores documents that will be requested by client processes across the network or the amount of memory available in system buffers which will be used for processing the server administrator.

Once instantiated. a servlet loops until the server is shut off or a destroy method is called on the servlet by the server Since the servlet operates in a continual loop as it waits for requests to act upon the server computer avoids the overhead of creating and destroying the servlet between requests to the servlet. In addition. keeping servlets alive between requests allows servlets to pass data and communicate amongst themselves For example servlets can maintain data about a user between sessions by the user This data can be shared among different servlets in order to customize a working environment within which the user works If servlets were created and destroyed on a per request basis. it would be much more difficult. if not practically impossible. for a servlet to understand the environment within which it runs and utilize this knowledge to provide improved processing capabilities The server computer can call a destroy method on the servlet when some resource limit in terms of time memory. etc is reached

The servlet application program interface (API) establisnes a standard for interfacing servlets with information servers. such as web servers The servlet API contains methods for initializing a servlet. processing the request. getting servlet information. and destroying the servlet. The servlet API allows platform independent servlets An example servlet interface is as follows

Servlet interface:

```
interface HttpServlet {
        Initialize (ServletContext, ServerProperties);
        Service(HttpRequest, HttpResponse);
        Destroy();
        }
```

The server computer passes objects that implement the "HttpRequest". while the servlet returns an "HttpResponse" object. The "ServletContext" interface is used to exchange information with the server environment. Some of the methods on the "ServletContext" object are "Getserver()" and "GetServlets()". "GetServer" returns a pointer to the parent server within which the instantiated Httpservlet runs. Using this pointer. the HttpServlet object can find out information about its parent server. The "GetServlet" method returns pointers to the servlets running on the parent server. The "ServerProperties" interface is used to exchange information regarding specific server properties established by a server administrator

Servlets support the familiar programming model of accepting requests and generating responses. The following is a simple servlet defining a single method called "service"

```
import java.servlet.*;
public class MyServlet extends GenericServlet {
        public void service (
                ServletRequest   request,
                ServletResponse   response
        ) throws ServletException, IOException
        {
        ...
        }
        ...
}
```

The service method is provided with Request and Response parameters. These parameters encapsulate the data sent by the client thereby allowing servlets to report status information, such as errors. Servlets normally retrieve most of their parameters through an input stream and send their responses using an output stream.

```
ServletInputStream in = request.getInputStream ()
ServletOutputStream out = response.getOutputStream ();
```

These input and output streams may be used with data in whatever format is appropriate. For example, an applet and servlet might exchange data using object serialization, HTML, or any number of image formats.

Since servlets are JAVA objects, they have instance-specific data. This means that in effect servlets are independent applications running within servers, without needing the complexity of additional classes (which are required by some alternative server extension APIs). Servlets have access to some servlet-specific configuration data at initialization time. This allows different instances of the same servlet class to be initialized with different data, and be managed as differently named servlets. The data provided at initialization time includes an area where each instance keeps its persistent instance-specific state.

Building upon the previous simple servlet examples, the following program code is an example of a servlet that is used to send Hypertext Markup Language (HTML) text when it is invoked:

BNSDOCID: <EP 0810524A1>

```
public class SimpleServlet extends GenericServlet  {
public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException
    {
            res.setContentType("text/html");
            PrintWriter out = new PrintWriter(res.getOutputStream());
            out.println("<HEAD><TITLE> SimpleServlet Output
</TITLE></HEAD><BODY>");
        out.println("<h1> SimpleServlet Output </h1>");
        out.println("<P>This is output from SimpleServlet.");
        out.println("</BODY>");
        out.flush();
    }
    public String getServletInfo() {
            return "A simple servlet";
    }
}
```

The following program code is an example of a servlet that uses the finger protocol to query information about users on specified host computers. The query string parameters < tt > user < /tt >. < tt > hosts < /tt >. and < tt > verbose < /tt > can be used to specify the user and hosts to query. The parameter <tt> user </tt> is the user name. <tt> hosts < /tt > is a comma-separated list of host names to query. and <tt> verbose </tt>. if specified. will cause verbose output to be generated. For example. <pre> http://goa/finger.html?user=dac&hosts=eno.doppio&verbose=yes </pre> will request full information about user "dac" on both hosts "eno" and "doppio".

```
public
class FingerServlet extends GenericServlet {
    /*
     * Port number for finger daemon.
     */
    static final int FINGER_PORT = 79;

    /**
     * Handles a single finger request from the client.
     */
    public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException
    {
        String user = req.getParameter("user");
        String hosts = req.getParameter("hosts");
        String verbose = req.getParameter("verbose");
        res.setContentType("text/html");

        PrintStream out = new PrintStream(res.getOutputStream());
        out.println("<html>");
        out.println("<head><title>Finger Servlet</title></head>");
        out.println("<body>");
        out.println("<h2>Finger results:</h2>");
        outprintln("<pre>");
        if (hosts == null) {
            finger(out, user, null, "yes".equalsIgnoreCase(verbose));
```


BNSDOCID: <EP 0810524A1>

```
                } else {
                        StringTokenizer st = new StringTokenizer(hosts, ",");
                        while (st.hasMoreTokens()) {
                                String host = st.nextToken();
                                out.println("[" + host + "]");
                                try {
                                        finger(out, user, host,
"yes".equalsIgnoreCase(verbose));
                                } catch (IOException e) {
                                        out.println(e);
                                }
                                out.println();
                        }
                }
                out.println("</pre>");
                out.println("</body></html>");
        }
        /*
         *      Sends finger output for a user and host to the specified output
         *      stream.
         */
        void finger(OutputStream out, String user, String host, boolean verbose)
                throws IOException
        {
                // open connection to finger daemon
                Socket s;
                if (host == null) {
                        s = new Socket(InetAddress.getLocalHost(),
FINGER_PORT);
                } else {
                        s = new Socket(host, FINGER_PORT);
                }
                // send finger command
```

```
                PrintStream ps = new PrintStream(s.getOutputStream());
                if (verbose) {
                        ps.print("/W");
                }
                if (user ! = null) {
                        ps.print(user);
                }
                ps.print("\r\n");
                ps.flush();
                // copy results to output stream
                InputStream in = s.getInputStream();
                byte[] buf = new byte [512];
                int len;
                while ((len = in.read(buf, 0, buf.length)) != -1) {
                        out.write(buf, 0, len);
                }
                s.close();
        }
}
```

Those skilled in the art will appreciate that servlets which are being used with the HTTP protocol may support any HTTP method, including GET, POST, HEAD, and more. They may redirect requests to other locations and send HTTP-specific error messages. They can get access to parameters which were passed through standard HTML forms, including the HTTP method to be performed and the URI, which identifies the destination of the request.

As indicated above, one of the biggest performance features of servlets is that they do not require creation of a new process for each request. In most environments, many servlets run in parallel within the same process as the server. When used in such environments with HTTP, servlets provide compelling performance advantages over both the CGI approach and the Fast-CGI approach. This is because servlets have a small computational expense during thread context switches. Since in most environments servlets can handle many client requests each time they are initialized, the cost of the initialization is spread over many methods. All the client requests to that service have the opportunity to share data and communications resources, benefitting more strongly from system caches.

Those skilled in the art will appreciate that the servlets embodying the invention can be used to dynamically extends Java-enabled servers. The servlets provide a general framework for services built using the request-response paradigm. The servlets can provide secure web-based access to data which is presented using HTML web pages and they can be used for interactively viewing or modifying that data using dynamic web page generation techniques.

The servlets embodying the invention may be used to provide customized multiuser services for customer bases. The servlets are also flexible enough to support standardized services, such as serving static web pages through the HTTP (or HTTPS) protocols, and proxying services. Since they are used for dynamic extensibility, they may be used in a plug-in style, supporting facilities such as search engines and semi-custom applications, such as web-based order entry or inventory systems.

Although the servlets are preferably written in JAVA, the servlet clients may be written in any language. When servlets are used in the middle tiers of distributed application systems, they can in turn be clients to other services, written in any language.

EPODOC: EP 0810524A1>

Those skilled in the art will appreciate that servlets may be used in several modes The basic mode is at the core of a request/response protocol. In addition servlets may be specialized to support protocols such as HTTP In HTTP based applications. servlets are portable. complete. and much more efficient replacement for CGI based extensions Also in HTTP applications. servlets may be used with HTML server side includes to dynamically generate part of a web document.

The foregoing description. for purposes of explanation. used specific nomenclature to provide a thorough understanding of examples of the invention However. it will be apparent to one skilled in the art that the specific details are not required in order to practice other examples of the invention. In other instances. well know circuits and devices are shown in block diagram form in order to avoid unnecessary distraction from the underlying principles Thus. the foregoing descriptions of specific embodiments of the present invention are presented for purposes of illustration and description

Claims

1. A method executed by a local server computer under the control of a program. said local server computer including a memory for storing said program said local server computer forming a portion of a client-server network. said method comprising the steps of.

   receiving a request from a client computer of said client-server network:
   determining that said request requires dynamically generated information from a servlet object of said client-server network:
   uploading from a remote server computer of said client-server network a specified servlet object corresponding to said request: and
   executing said specified servlet object to obtain dynamically generated information corresponding to said request.

2. The method of claim 1 further comprising the step of passing dynamically generated information from said specified servlet object to a web server operating on said local server computer. said passing step being facilitated with an application program interface.

3. The method of claim 2 wherein said application programming interface specifies techniques for performing at least one of the following operations: initializing a servlet object. executing a servlet object. and destroying a servlet object.

4. The method of claim 2 wherein said specified servlet object and said application program interface are specified as object bytecodes in the JAVA programming language

5. The method of claim 2 further comprising the step of sending said dynamically generated information from said web server to said client computer.

6. The method of claim 1 wherein said executing step includes the steps of

   executing said specified servlet in a security area of said local server computer: and
   passing said dynamically generated information from said security area to a non-security area of said local server computer.

7. The method of claim 1 wherein said local server computer stores a plurality of servlet objects. each of said servlet objects continuously operating until invoked in response to a specified request from a client computer.

8. The method of claim 7 wherein said plurality of servlet objects pass data to one another.

9. The method of claim 7 wherein said selected servlet objects of said plurality of servlet objects are instantiated at the start-up of said local server computer.

10. The method of claim 7 wherein selected servlet objects of said plurality of servlet objects are instantiated in response to a demand from said client computer.

11. The method of claim 7 wherein selected servlet objects of said plurality of servlet objects are instantiated in response to an activated servlet URL

12. The method of claim 11 wherein said servlet URL includes arguments

13. The method of claim 1 wherein said receiving step includes the step of storing said request in a connection queue

14. The method of claim 13 wherein said determining step includes the step of selecting a handler thread from a pool of handler threads to execute said determining step

15. The method of claim 14 further comprising the step of operating said pool of handler threads by selectively creating a new handler thread and destroying an old handler thread

16. The method of claim 15 wherein said operating step includes the step of reusing a buffer memory space of said old handler thread for said new handler thread

17. A computer readable memory that can be used to direct a server computer of a client-server computer network to function in a specified manner, comprising

a first set of instructions to receive a request from a client computer of said client-server network,
a second set of instructions to determine that said request requires dynamically generated information from a servlet object of said client-server network;
a third set of instructions to upload from a remote server computer of said client-server network a specified servlet object corresponding to said request, and
a fourth set of instructions to execute said specified servlet object to obtain dynamically generated information corresponding to said request

18. The apparatus of claim 17 further comprising a fifth set of instructions to pass through an application program interface, dynamically generated information from said specified servlet object to a web server operating on said local server computer.

19. The apparatus of claim 18 further comprising a sixth set of instructions to pass said dynamically generated information from said web server to said client computer

20. The apparatus of claim 17 further comprising a seventh set of instructions to store a plurality of servlet objects on said server computer each of said servlet objects continuously operating until invoked in response to a specified request from a client computer.

21. The apparatus of claim 20 wherein said seventh set of instructions include instructions to pass data between said plurality of servlet objects.

22. A computer readable memory that can be used to direct a server computer of a client-server computer network to function in a specified manner, comprising:

a first set of instructions to receive a request from a client computer of said client-server computer network,
a second set of instructions to determine that said request requires dynamically generated information from a servlet object of said server computer;
a third set of instructions to execute said specified servlet object to obtain dynamically generated information corresponding to said request; and
a fourth set of instructions to pass said dynamically generated information to said client computer.

23. The apparatus of claim 22 wherein said second set of instructions include instructions to interpret a servlet URL corresponding to said request.

24. The apparatus of claim 22 wherein said second set of instructions include instructions to interpret a servlet URL with arguments.

25. The apparatus of claim 22 further comprising a fifth set of instructions to store a plurality of servlet objects on said

EP 0 810 524 A1

server computer, each of said servlet objects continuously operating until invoked in response to a specified request from a client computer

26. The apparatus of claim 20 wherein said fifth set of instructions include instructions to pass data between said plurality of servlet objects.

27. A client-server computer network comprising:

a client computer to generate a request; and
a server computer to

determine that said request requires dynamically generated information from a servlet object of said server computer,
execute said specified servlet object to obtain dynamically generated information corresponding to said request, and
pass said dynamically generated information to said client computer.

28. The apparatus of claim 27 further comprising a remote server computer storing a set of servlet objects that can be passed to said server computer.

29. The apparatus of claim 27 wherein said server computer stores a plurality of servlet objects, each of said servlet objects continuously operating until invoked in response to a specified request from said client computer

30. The apparatus of claim 29 wherein said plurality of servlet objects pass data between themselves.

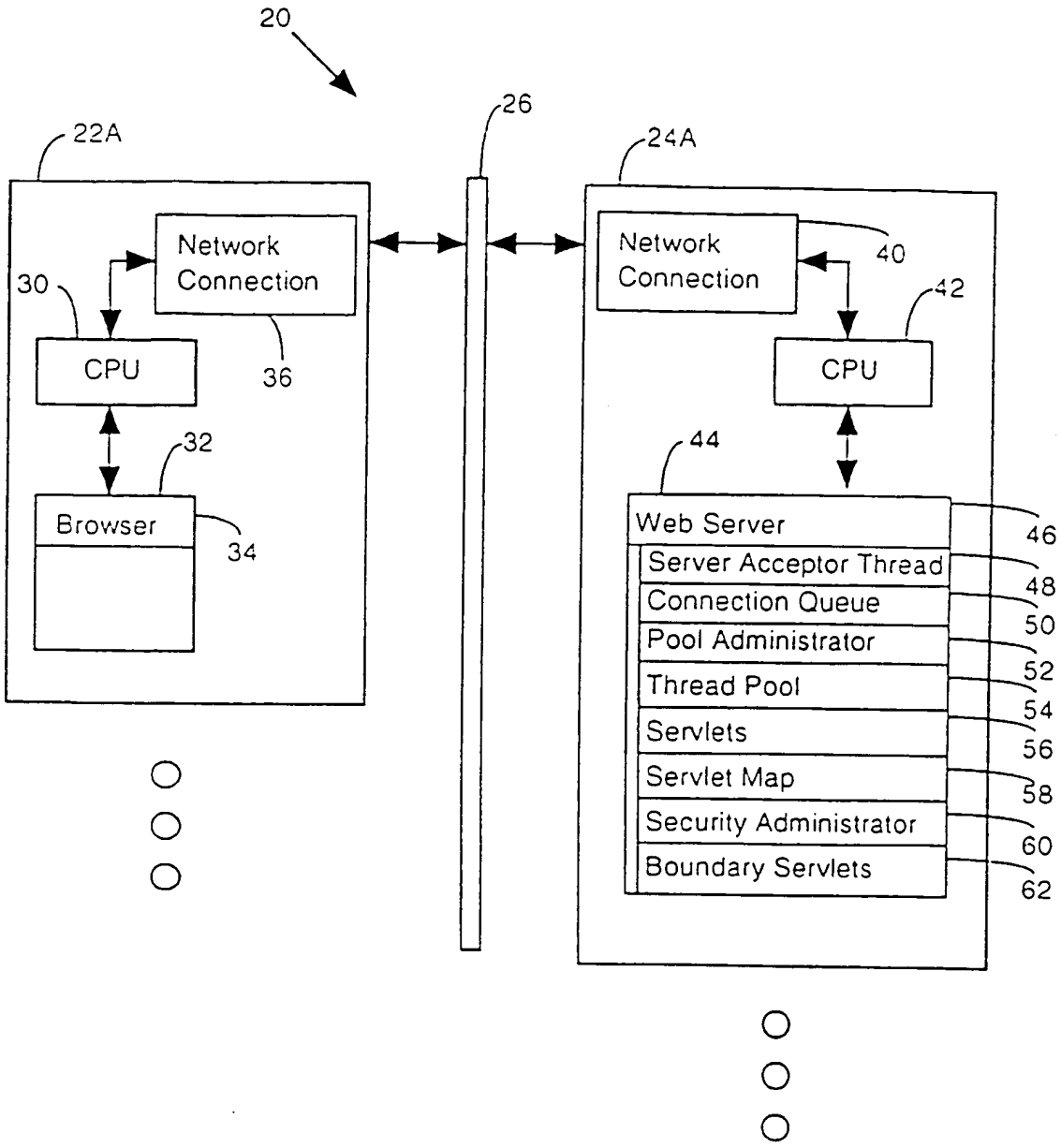

Figure 1

BNSDOCID: <EP 0810524A1>

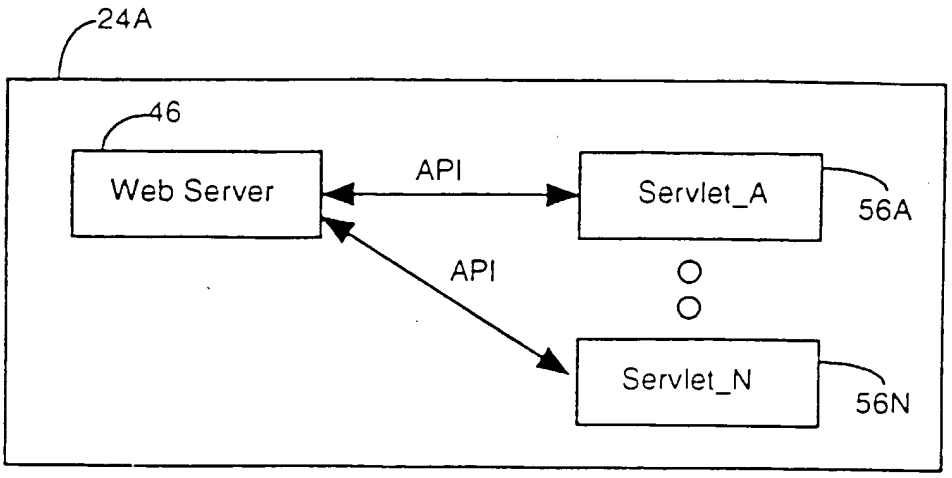

Figure 2

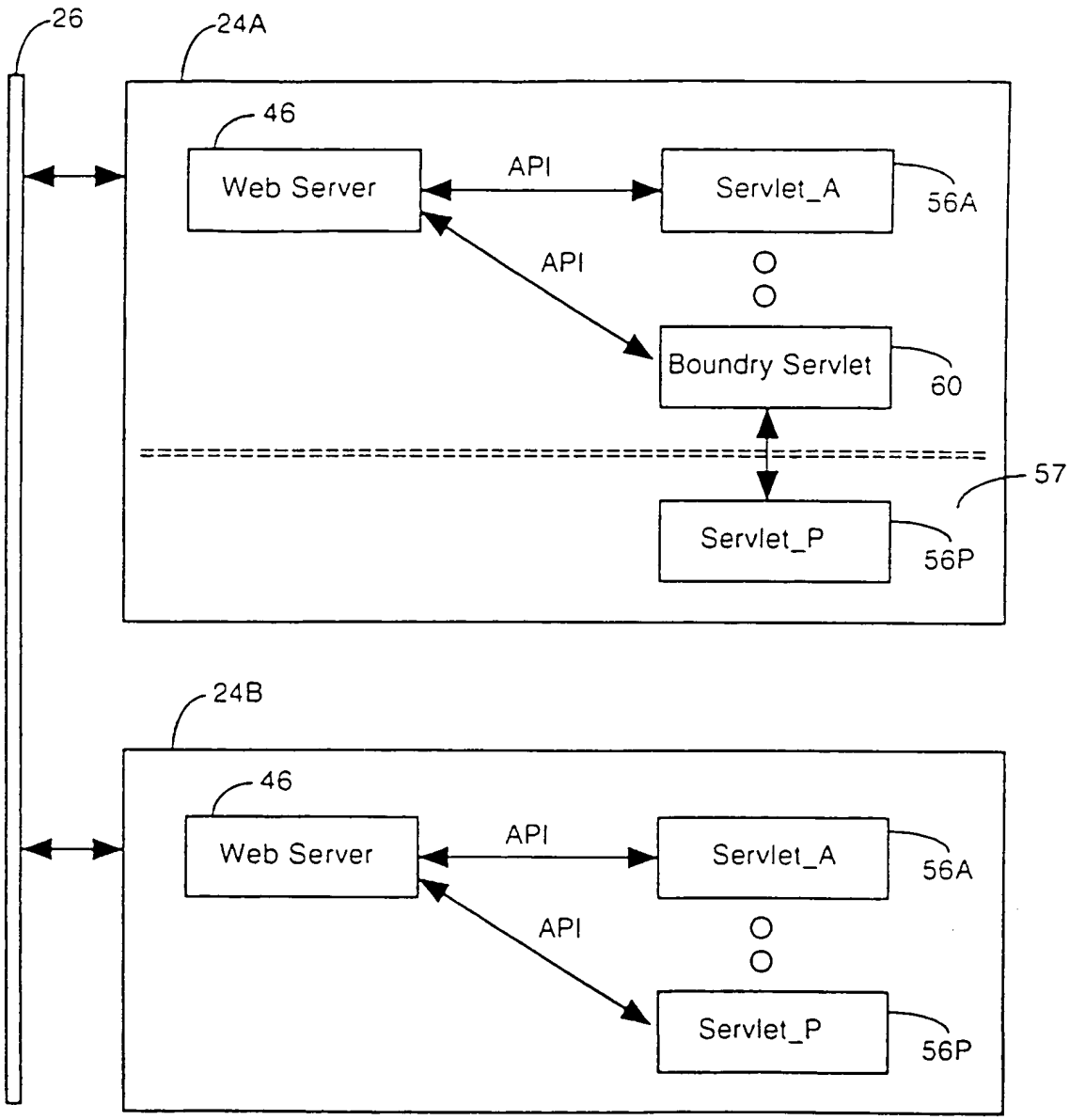

Figure 3

BNSDOCID: <EP 0810524A1>

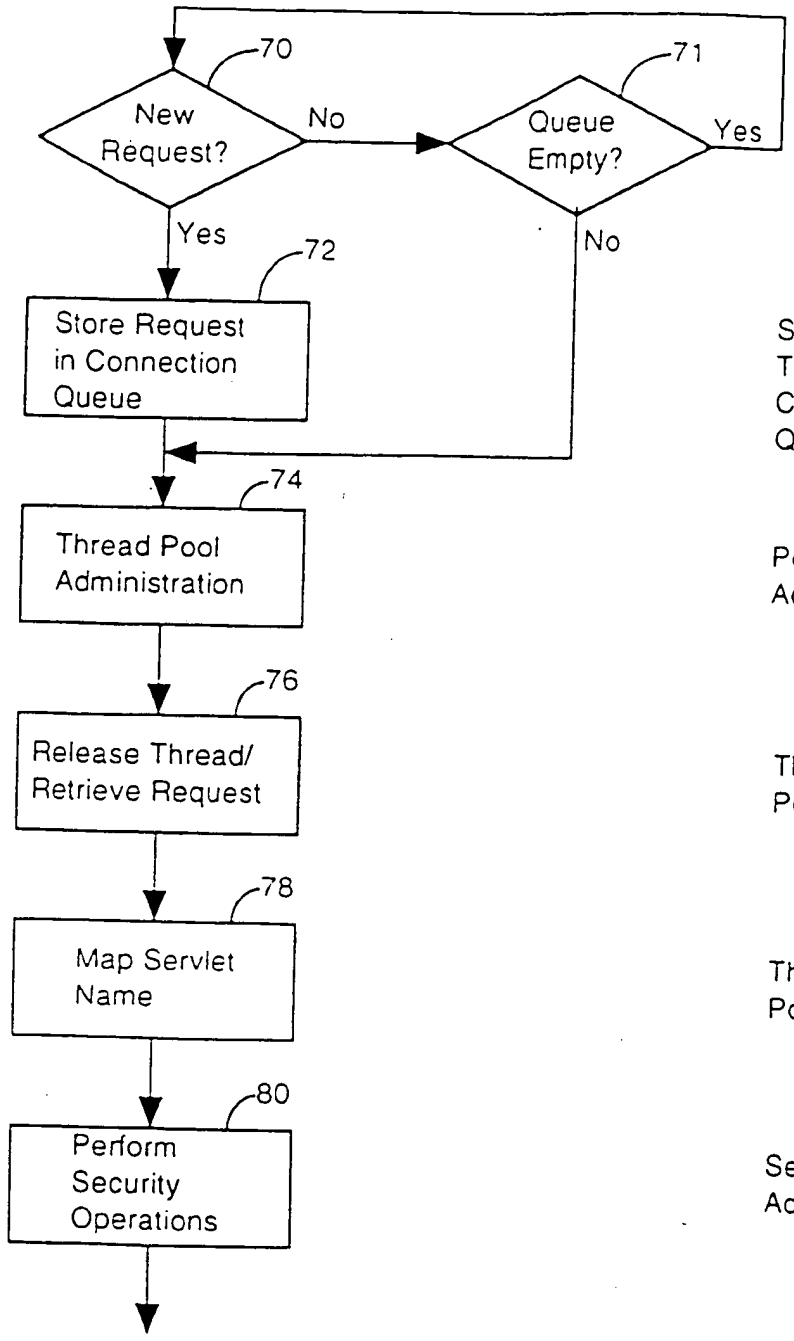
70
New Request?
No
71
Queue Empty?
Yes
Yes
No
72
Store Request in Connection Queue
74
Thread Pool Administration
76
Release Thread/ Retrieve Request
78
Map Servlet Name
80
Perform Security Operations
(To Step 82; Figure 5)
Server Acceptor Thread (48)
Server Acceptor Thread (48)/ Connection Queue (50)
Pool Administrator (52)
Thread from Thread Pool(54)
Thread from Thread Pool (54)/Servlet Map (58)
Security Administrator (60)

Figure 4

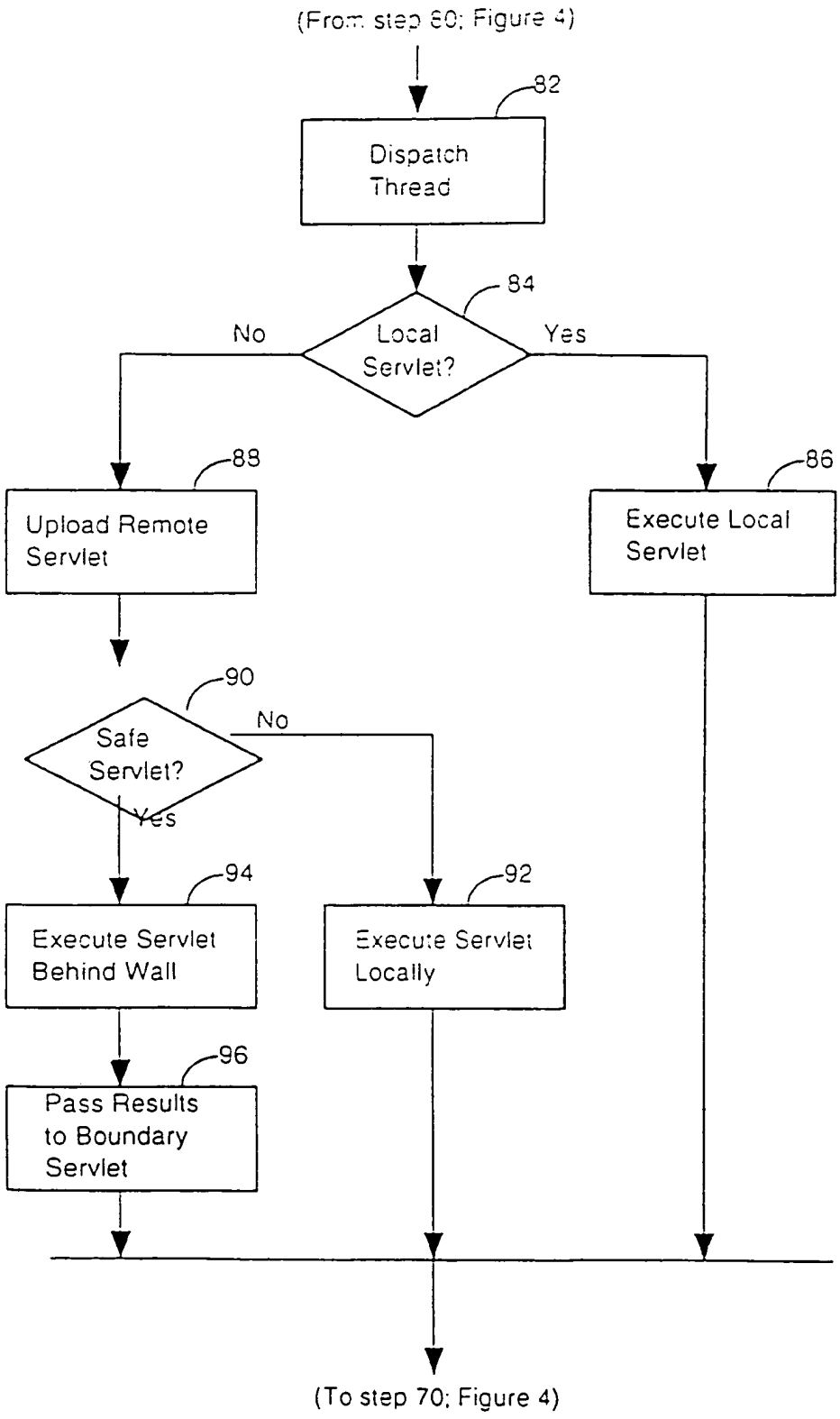

Figure 5

BNSDOCID: <EP 0810524A1>

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 97 30 3576

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
P,X	NETSCAPEWORLD, 5 May 1997, XP002041555 JIM LOWE: "How Java servlets can replace CGI scripts - for ease, performance & more" (http://www.cs.berkeley.edu/~padmanab/papers/masters-tr.ps) (http://www.netcapeworld.com/netscapeworld/nw-05-1997/nw-05-bytecode.html) * the whole document * ---	1-30	G06F9/46
P,X	NETSCAPEWORLD, - 4 July 1996 XP002041556 TRISHA GORMAN: "Server-side applets in Java generate developer anticipation" (http://www.netscapeworld.com/netscapeworld/nw-07-1996/nw-07-jeeves.html) * the whole document * ---	1-30	
A	UNIVERSITY OF CALIFORNIA AT BERKELEY REPORT NO. UCB/CSD-95-875, May 1995, COMPUTER SCIENCE DIVISION, pages 1-24, XP002041557 VENKATA N. PADMANABHAN: "Improving World Wide Web Latency" (http://www.cs.berkeley.edu/~padmanab/papers/masters-tr.ps) * abstract * * page 8, paragraph 2 - paragraph 3 * --- -/--	1,17,22, 27	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	23 September 1997	Fonderson, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 97 30 3576

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	FIFTH INTERNATIONAL WORLD WIDE WEB CONFERENCE (POSITION PAPER FOR THE WORKSHOP "PROGRAMMING THE WEB - A SEARCH FOR API'S", 6 May 1996, PARIS, FRANCE, XP002041558 MARK R. BROWN: "FastCGI: A High-Performance Gateway Interface" (http://www.fastcgi.com/kit/doc/www5-api-workshop.html) * the whole document * ---	1,17,22, 27	
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 38, no. 5, May 1995, NEW YORK US, pages 199-200, XP002041559 "Control of Dynamic Threads Pool for Concurrent Remote Procedure Calls" * the whole document * -----	14,15	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	23 September 1997	Fonderson, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

BNSDOCID: <EP 0810524A1>

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ❑ BLACK BORDERS
- ❑ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ❑ FADED TEXT OR DRAWING
- ❑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ❑ SKEWED/SLANTED IMAGES
- ❑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ❑ GRAY SCALE DOCUMENTS
- ❑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ❑ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ❑ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE LEFT BLANK